APPENDIX

All methods return HRESULTs unless otherwise stated.

IMediaIndexManager

OpenScheme(LPCWSTR wszName, void* pReserved, LPMEDIAINDEXSCHEME* ppScheme)

Opens a specific scheme that has already been registered under the name *wszName*. A scheme object is returned in the *ppScheme* parameter. A parameter *pReserved* is reserved for future use.

The usual flow would be to try to open a named scheme, and if this method returns a failure code, register the scheme and try again.

RegisterSchemeFromXMLObject(LPCWSTR wszName, IXMLDOMDocument* pXMLDoc)

Takes an XML document object identified by *pXMLDoc* and stores it in the registry under the specified name *wszName*. Schemes are stored in:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\MediaContentIndex\Schemes

RegisterSchemeFromXMLScript(LPCWSTR wszName, LPCWSTR wszXMLScript)

Takes an XML document in string form identified by *wszXMLScript* and stores it in the registry under the specified name *wszName*, after validating that it is a createable XML document. Schemes are stored in:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\MediaContentIndex\Schemes

Basically, this method just CoCreates a new XML document object and calls RegisterSchemeFromXMLObject if the XML document is valid.

RegisterSchemeFromXMLFile(LPCWSTR wszName, LPCWSTR wszSchemeFilePath)

Takes an XML document in string form (stored in the specified file *wszSchemeFilePath*) and stores it in the registry under te specified name *wszName*, after validating that it is a createable XML document. Schemes are stored in:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\MediaContentIndex\Schemes

Basically, this method just CoCreates a new XML document object and calls RegisterSchemeFromXMLObject if the XML document is valid.

BeginSchemeEnumeration(void)

Tells the index that we want to begin getting a list of schemes that are registered.

Returns: E_MM_NOSCHEMES if no schemes are found in the registry.

EnumerateScheme(LPWSTR wszName, DWORD cchName)

Gets the next registered scheme's name after *wszName*. The name parameter *cchName* is filled in up to the count of characters provided (note that this is a count of UNICODE characters available, not the number of bytes in the string).

Returns: E_MM_NOENUMERATION if BeginSchemeEnumeration() has not been called successfully.

EndSchemeEnumeration(void)

Tells the index that we're done enumerating scheme names. Can be called safely even if BeginSchemeEnumeration() has not been called.

IMediaIndexScheme

GetSchemeInfo (LPWSTR wszSchemeName, IXMLDOMDocument** ppXMLDoc)

Returns the name of the current scheme in *wszSchemeName* and its XML document object in *ppXMLDoc*. This object must be released by the caller.

OpenIndex (void)

Opens the Media Index. Must be called in order to get a root. If the Media Index is already open, calling this routine again is an error. Opening the index is a time-consuming operation and therefore opening and closing the index multiple times should be avoided.

Returns: E_MM_DBALREADYOPEN if OpenIndex has already been called.
E_MM_BADSCHEME if we can't extract a file name from the XML scheme.

CloseIndex (void)

Closes the Media Index. Must be called when exiting the client application, otherwise there will be leaks and the database may be left in an inconsistent state.

Returns: E_MM_DBNOTOPEN if OpenIndex() has not been called successfully.

BeginUserEnumeration (void)

Tells the index that we want a list of users.

Returns: E_MM_DBNOTOPEN if OpenIndex() has not been called successfully.

EnumerateUser (LPWSTR wszUserName)

Returns the name of a user in *wszUserName*, and moves an internal pointer to the next one.

Returns: E_MM_NOENUMERATION if BeginUserEnumeration has not been called successfully.
E_MM_EOL if there are no more users to enumerate.

EndUserEnumeration (void)

Tells the index that we're done getting our list of users. Can be called safely even if BeginUserEnumeration() has not been called.

SetCurrentUser (LPCWSTR wszUserName)

Sets a user name as the name identified in *wszUserName*. Note that this is not necessary if you wish to have the current logged-on user as the current user of the index, since this is the default behavior.

Returns: E_MM_DBNOTOPEN if OpenIndex() has not been called successfully.

BeginTransaction (void)

Begins a transaction in the index. A transaction is a logical set of operations in the database that can be applied as one set and canceled if there are any failures. For example, if you wish to enter an album and all of its properties, but don't want any of the information saved if there is a failure during the process, you would begin a transaction before doing anything else.

Returns: E_MM_DBNOTOPEN if OpenIndex() has not been called successfully.

EndTransaction (BOOL fCommit)

Ends the current transaction and commits the data to the database, if *fCommit* is returned as true. Otherwise, it throws away all database operations since "BeginTransacation" was called and returns *fCommit* as false.

Returns: E_MM_DBNOTOPEN if OpenIndex() has not been called successfully.

GetRoot (LPMEDIAINDEXROOT* ppObject)

Returns the index root in *ppObject*.

Returns: E_MM_DBNOTOPEN if OpenIndex() has not been called successfully.

GetIndexFilename (LPWSTR wszFilename)

Returns the filename of the index associated with this scheme in *wszFilename*.

Returns: E_INVALIDARG if wszFilename is NULL.

BeginObjectTypeEnumeration (LPDWORD pdwCount)

Starts enumerating object types from the current scheme. Unlike actually enumerating objects, this can be done even when there are no instances of the object in the data base. Caller can ask for an optional count of objects through the *pdwCount* parameter.

Returns: E_MM_ENUMERATIONACTIVE if you're already enumerating object types for this scheme.

EnumerateObjectType (REFGUID guidObjectID, LPWSTR wszObjectName)

Returns the next object type in the enumeration. Both parameters get values returned: the *guidObjectID* is the GUID of the object type, and the *wszObjectName* is the name of the object type from the scheme.

Returns: E_MM_NOENUMERATION if you haven't called BeginObjectTypeEnumeration successfully.
E_MM_EOL if there are no more object types.

EndObjectTypeEnumeration (void)

Required call to end an object type enumeration. May be safely called even though BeginObjectTypeEnumeration() was not.

BeginPropertySetTypeEnumeration (LPCGUID pguidObjectID, LPDWORD pdwCount)

Starts enumerating property set types from the current scheme. Unlike actually enumerating property sets, this can be done even when there are no instances of the property set in the data base. If the optional *pguidObjectID* parameter is specified, this will only enumerate property sets for that object; otherwise, this enumerates all property sets in scheme. Caller can ask for an optional count of property sets through the *pdwCount* parameter.

Returns: E_MM_ENUMERATIONACTIVE if you're already enumerating property set types for this scheme.
E_OUTOFMEMORY if internal allocation failure.

EnumeratePropertySetType (REFGUID guidObjectID, REFGUID guidPropertySetID, LPDWORD pdwPropertySetID, LPWSTR wszPropertySetName)

Returns the next property set type in the enumeration. All four parameters get values returned (if they're specified): *guidObjectID* is the GUID of the object type to which the property set belongs, *guidPropertySetID* is the GUID of the property set type itself, *pdwPropertySetID* is the ordinal of the property set in the data base, and *wszPropertySetName* is the name of the property set type from the scheme.

Returns: E_MM_NOENUMERATION if you haven't called BeginPropertySetTypeEnumeration successfully.
E_MM_EOL if there are no more property set types.

EndPropertySetTypeEnumeration (void)

Required call to end a property set type enumeration. May be safely called even though BeginPropertySetTypeEnumeration() was not.

BeginPropertyTypeEnumeration (LPCGUID pguidPropertySetID, LPDWORD pdwCount)

Starts enumerating property types from the current scheme. Unlike actually enumerating properties, this can be done even when there are no instances of the property in the data base. If the optional *pguidPropertySetID* parameter is specified, this will only enumerate properties for that property set; otherwise, this enumerates all properties in the scheme. Caller can ask for an optional count of properties through the *pdwCount* parameter.

Returns: E_MM_ENUMERATIONACTIVE if you're already enumerating property types for this scheme.
E_OUTOFMEMORY if internal allocation failure.

EnumeratePropertyType (REFGUID guidPropertySetID, LPDWORD pdwPropertyID, LPWSTR wszPropertyName, LPDWORD pdwDataType)

Returns the next property type in the enumeration. All four parameters get values returned (if they're specified): *guidPropertySetID* is the GUID of the property set type to which the property belongs, *pdwPropertyID* is the ordinal of the property type in the data base, *wszPropertyName* is the name of the property type from the scheme, and *pdwDataType* is the data type of the property value.

Returns: E_MM_NOENUMERATION if you haven't called BeginPropertyTypeEnumeration successfully.
E_MM_EOL if there are no more property types.

EndPropertyTypeEnumeration (void)

Required call to end a property type enumeration. May be safely called even though BeginPropertyTypeEnumeration() was not.

WriteSchemeToXMLFile (LPCWSTR wszSchemeFile)

Exports the XML data for the scheme as a string to the file path specified as *wszSchemeFile.*

Returns: E_MM_REGNOKEY or E_MM_REGNOVALUE if scheme wasn't found in registry.

E_MM_CANTCREATEFILE or E_MM_CANTWRITEFILE if there were problems exporting to the specified file path.

IMediaIndexRoot

BeginObjectEnumeration (LPCGUID pguidObjectType, DWORD dwStartingIndex, BOOL fRestrictToCurrentUser, LPDWORD pdwCount)

Tells the index that we want to start getting a list of objects. The parameters are:

pguidObjectType: NULL or a pointer to a GUID. NULL means "all objects" and a valid GUID means "just objects of this type" (for example, just Albums).

dwStartingIndex: Start at this index, 0 is the beginning.

fRestrictToCurrentUser: Reserved for future use.

pdwCount: If this is non-NULL, returns the count of objects that will be enumerated. This can be an expensive operation, so unless you really need the count before enumerating, don't use this parameter.

EnumerateObjects (LPMEDIAINDEXOBJECT *ppObject)

Returns as *ppObject* the next object in the enumeration. Will return a failed hresult when there is no more data.

Returns: E_MM_NOENUMERATION if BeginObjectEnumeration() has not been called successfully.

E_MM_EOL if there are no more objects to enumerate.

EndObjectEnumeration (void)

Tells the index that we're done with the current enumeration. Can be called safely even if BeginObjectEnumeration() has not been called.

Search (LPCGUID pguidObjectTypeList, DWORD dwObjTypes, LPCGUID pguidPropertySetTypeList, WORD dwPropertySetTypes, LPDWORD pdwPropertyNumberList, DWORD dwProperties, LPCWSTR wszSearch, DWORD dwFlags, LPMEDIAINDEXSEARCHSINK pResultSink)

Searches for properties that match all specified object types, property set types and property ID's, then compares the value of each qualifying property against the specified search string. Returns any matches and final status to the object specified in *pResultSink.* Search is done on a separate thread, and matches are returned asynchronously, as they are found. It is the client's responsibility to order the list of matches in a logical way.

Returns: E_MM_SEARCHINPROGRESS if there is an active search on this root.

The parameters are:

pguidObjectTypeList: pointer to array of object type GUID's to match.
dwObjTypes: count of entries in *pguidObjectTypeList.*
pguidPropertySetTypeList: pointer to array of property set type GUID's to match.
dwPropertySetTypes: count of entries in *pguidPropertySetTypeList.*
pdwPropertyNumberList : pointer to array of property ID's to match.
dwProperties: count of entries in *pdwPropertyNumberList.*
wszSearch: string to match against property value of qualifying properties.
dwFlags: flags governing the way the search and string matching is performed.
pResultSink: pointer to object that will receive the search results, as they are found.

CancelSearch()

Can be called to cancel an active search.

Returns: E_MM_UNINIT if Search() has not been called successfully.

CreateObject (REFGUID guidObjectType, LPMEDIAINDEXOBJECT* ppObject)

Creates a new object of the type specified in *guidObjectType*, and returns the new object in *ppObject.*

Returns: E_MM_NOSUCHOBJ if guidObjectType not found.

FetchObject (REFGUID guidObjectType, DWORD dwInstanceID, LPMEDIAINDEXOBJECT* ppObject)

Fetches the specific object of the type *guidObjectType* and instance ID *dwInstanceID* specified, returning the object as *ppObject.*

Returns: E_MM_NOSUCHOBJ if guidObjectType not found.
E_MM_EOL if dwInstanceID not found.

FetchObjectByNumericProperty (REFGUID guidObjectType, REFGUID guidPropertySetID, DWORD dwPropertyID, DWORD dwValue, LPMEDIAINDEXOBJECT* ppObject)

Fetches a specific object of the type *guidObjectType* and property set *guidPropertySetID* based on the numeric value *dwValue* of a specific property *dwPropertyID*, returning the object as *ppObject*. This is valuable if you know that the numeric property is a "key" of the object (for example, the Album Hash ID for a CD).

Returns: E_MM_NOSUCHOBJ if guidObjectType not found.
E_MM_NOSUCHPROPSET if guidPropertySetID not found.
E_MM_NOSUCHPROP if dwPropertyID not found.

FetchObjectByTextProperty (REFGUID guidObjectType, REFGUID guidPropertySetID, DWORD dwPropertyID, LPCWSTR wszSearch, LPMEDIAINDEXOBJECT* ppObject)

Fetches a specific object of the type *guidObjectType* and property set *guidPropertySetID* based on its text property *dwPropertyID* having the value *wszSearch*. This is valuable if you know that the text property is a "key" of the object (for example, the name property of an artist).

Returns: E_MM_NOSUCHOBJ if guidObjectType not found.
E_MM_NOSUCHPROPSET if guidPropertySetID not found.
E_MM_NOSUCHPROP if dwPropertyID not found.

RemoveObject (LPMEDIAINDEXOBJECT pObject)

Removes an object identified by *pObject* from the index. This clears all of its properties and relationships.

Returns: E_MM_NOSUCHOBJ if object not found in object values table.

SetSearchExclusions (LPCWSTR wszExclude)

Optionally allows specification of a list of words *wszExclude* that will be ignored from the user's input, only when matching on individual words (i.e. no words are excluded when matching on phrases). Generally includes things like "the", "a", and "and". Format of the wszExclude string is that each word is terminated with a NULL, and the entire string is terminated with an extra NULL (so the string must end in 2 NULL's).

Added to allow localization support for excluded words.

SetSearchSeparators (LPCWSTR wszSep)

Optionally allows specification of the list of characters *wszSep* that are deemed to mark the start/end of a word. When parsing the *wszSearch* string of the Search() method, and any strings in the data base, these separators allow the Media Index to determine where words start and end, so that individual words in a phrase may be compared.

If no separator list is supplied by the client, the separator list defaults to a single space band.

Added to allow localization support for separator characters.

IMediaIndexObject

GetObjectInfo (REFGUID guidObjectType, LPWSTR wszObjectTypeName, LPWSTR wszObjectDisplayName, LPDWORD pdwInstanceID)

Returns the GUID (*guidObjectType* is an OUT parameter) and instance ID (*pdwInstanceID*) of this object.

Optional name parameters return the object name *wszObjectTypeName* from the scheme, and the "display name" *wszObjectDisplayName* property of the object (if any).

BeginConnectionEnumeration (BOOL fChildren, DWORD dwStartingIndex, LPCGUID pguidObjectType, BOOL fRestrictToCurrentUser, LPDWORD pdwCount)

Tells the index that we want to start getting a list of objects. The parameters are:

fChildren: TRUE to enumerate children, FALSE to enumerate parents.

pguidObjectType: NULL or a pointer to a GUID. NULL means "all objects" and a valid GUID means "just objects of this type" (for example, just Albums).

dwStartingIndex: Start at this index, 0 is the beginning.

fRestrictToCurrentUser: Reserved for future use.

pdwCount: If this is non-NULL, returns the count of objects that will be enumerated. This can be an expensive operation, so unless you really need the count before enumerating, don't use this parameter.

EnumerateConnection (LPMEDIAINDEXOBJECT *ppObject)

Returns the next object in the enumeration as *ppObject*. Will return a failed hresult when there is no more data.

Returns: E_MM_NOENUMERATION if BeginConnectionEnumeration() has not been called successfully.
E_MM_EOL if no more connections to enumerate.

EndConnectionEnumeration (void)

Ends the connection enumeration. Can be called safely even if BeginConnectionEnumeration() has not been called.

AddChild (LPMEDIAINDEXOBJECT pAddObject, LPMEDIAINDEXOBJECT pNextObject)

Adds a child to this object. *pAddObject* is the new child, *pNextObject* is the object that will come next in the list of children, or NULL to put the new child at the end of the list.

RemoveChild (LPMEDIAINDEXOBJECT pObject)

Removes the child *pObject* from the objects list of relationships.

AddPropertyType (REFGUID guidPropertySetID, DWORD dwPropertyID, LPCWSTR wszPropertyName, DWORD dwDataType, LPMEDIAINDEXPROPERTYVALIDATION pValid)

Allows the caller to define a new property type for a "user-defined" property set *guidPropertySetID*. The new property value is identified in *dwPropertyID*, the name of the new property is *wszPropertyName*, the type of data the property stores is identified by *dwDataType*, and the validation parameters are identified by *pValid*. This is the way that properties may be dynamically added to a scheme. The *pValid* parameter is optional, and the *wszPropertyName* parameter can be NULL (but supplying a name is strongly recommended).

Returns: E_MM_DBNOTOPEN if data base not supplied in constructor.
E_MM_PROPEXISTS if specified property set ID / property ID pair already exist.
E_MM_NOSUCHPROPSET if specified property set ID doesn't exist .
E_MM_CANTADDPROPERTY if specified property set exists, but isn't a "user" property set.

GetPropertyType (REFGUID guidPropertySetID, DWORD dwPropertyID, LPWSTR wszPropertyName, DWORD *pdwDataType, LPMEDIAINDEXPROPERTYVALIDATION *ppValid)

Given the first two parameters identifying a property set *guidPropertySetID* and property type *dwPropertyID*, returns the next three parameters from the data base: the name of the property *wszPropertyName*, the type of data stored by the property *pdwDataType*, and optionally the validation parameters *ppValid*.

Returns: E_MM_DBNOTOPEN if data base not supplied in constructor.
E_INVALIDARG if 'wszPropertyName' or 'pdwDataType' are NULL.
E_MM_NOSUCHPROP if specified property type not found.
E_OUTOFMEMORY if an internal allocation failed.

SetProperty (REFGUID guidPropertySetID, DWORD dwPropertyID, LPCWSTR wszPropertyName, DWORD dwDataType, LPCVOID pvBuffer, DWORD dwBufferSize)

Validates and adds/modifies specified property instance (specified by the property set *guidPropertySetID* and the property type *dwPropertyID*), with the specified characteristics (the name of the property *wszPropertyName*, the type of data stored in the property *dwDataType*, and the data identified by *pvBuffer* of size *dwBufferSize*.

Returns: E_MM_DBNOTOPEN if data base not supplied in constructor.
E_INVALIDARG if *pvBuffer* is NULL.
E_MM_NOSUCHPROP if specified property instance (*guidPropertySetID* / *dwPropertyID*) doesn't exist.
E_MM_INVALIDPROP if specified values don't pass validation as indicated in the scheme.

GetProperty (REFGUID guidPropertySetID, DWORD dwPropertyID, LPWSTR wszPropertyName, LPDWORD pdwDataType, LPVOID pvDataBuffer, DWORD dwBufferSize, LPDWORD pdwRequiredSize)

Returns data (the property name *wszPropertyName*, the type of data stored in the property *pdwDataType*) about the specified property instance (specified by the property set *guidPropertySetID* and the property type *dwPropertyID*) into a buffer (identified by *pvDataBuffer* of size *dwBufferSize*). The parameters *pdwDataType*, *pvDataBuffer* and *pdwRequiredSize* are optional return parameters and may be left NULL.

Returns: E_MM_DBNOTOPEN if data base not supplied in constructor.
E_MM_NOSUCHPROP if specified property instance (*guidPropertySetID* / *dwPropertyID*) doesn't exist.

RemoveProperty (REFGUID guidPropertySetID, DWORD dwPropertyID)

Removes specified property instance (specified by the property set *guidPropertySetID* and the property type *dwPropertyID*) from the data base.

Returns: E_MM_DBNOTOPEN if data base not supplied in constructor.
E_MM_NOSUCHPROP if specified property instance (*guidPropertySetID* / *dwPropertyID*) doesn't exist.

CreatePropertyValidation (LPMEDIAINDEXPROPERTYVALIDATION *ppValid)

Creates a new instance of a property validation object *ppValid* in memory. Used to populate a property validation object before passing it to other API's. Caller must Release() returned pointer when it is no longer needed.

Returns: E_POINTER if 'ppValid' is NULL.
E_OUTOFMEMORY if internal allocation failed.

SetPropertyValidation (REFGUID guidPropertySetID, DWORD dwPropertyID, LPMEDIAINDEXPROPERTYVALIDATION pValid)

Assign validation parameters *pValid* to an existing property (specified by the property set *guidPropertySetID* and the property type *dwPropertyID*). May override existing validation data for the property.

Returns: E_MM_DBNOTOPEN if data base not supplied in constructor.
E_MM_NOSUCHPROPSET if specified property set doesn't exist.
E_MM_NOSUCHPROP if specified property instance (*guidPropertySetID* / *dwPropertyID*) doesn't exist.

GetPropertyValidation (REFGUID guidPropertySetID, DWORD dwPropertyID, LPMEDIAINDEXPROPERTYVALIDATION *ppValid)

Get validation parameters *ppValid* for an existing property (specified by the property set *guidPropertySetID* and the property type *dwPropertyID*). Caller must Release() returned pointer when it is no longer needed.

Returns: E_MM_DBNOTOPEN if data base not supplied in constructor.
E_MM_NOSUCHPROPSET if specified property set doesn't exist.
E_MM_NOSUCHPROP if specified property instance (*guidPropertySetID* / *dwPropertyID*) doesn't exist.

BeginPropertyEnumeration (LPCGUID pguidPropertySetID, LPDWORD pdwCount)

Start enumerating actual property instances for the property set specified as *pguidPropertySetID*. If no property set is specified, enumerates all properties. Returns an optional count of properties that will be enumerated in *pdwCount*.

EnumerateProperty (REFGUID guidPropertySetID, LPDWORD pdwPropertyID, LPWSTR wszPropertyName, LPDWORD pdwDataType, LPVOID pvDataBuffer, DWORD dwBufferSize, LPDWORD pdwRequiredSize)

Returns data (the property name *wszPropertyName*, the type of data stored in the property *pdwDataType* and the number of bytes required to store the property *pdwRequiredSize*) about the current property instance (specified by the property set *guidPropertySetID* and the property type *pdwPropertyID*) in the data base into a buffer (identified by *pvDataBuffer* of size *dwBufferSize*).

Returns: E_MM_NOENUMERATION if BeginPropertyEnumeration() has not been called successfully.
E_MM_EOL if no more properties to enumerate.

EndPropertyEnumeration (void)

Ends the property enumeration. Can be called safely even if BeginPropertyEnumeration() has not been called.

IsParentOf (LPMEDIAINDEXOBJECT pObject)
Returns a BOOL value indicating whether the IMediaIndexObject that has the IsParentOf function as a member is a parent of *pObject*.

IsChildOf (LPMEDIAINDEXOBJECT pObject)
Returns a BOOL value indicating whether the IMediaIndexObject that has the IsChildOf function as a member is a child of *pObject*.

GetPropertySetNameFromGUID (REFGUID guidPropertySetID, LPWSTR wszPropertySetName)
Given a GUID of a property set type *guidPropertySetID*, return its type name *wszPropertySetName* from the scheme.
The property set name could alternativley be returned as an optional property of EnumerateProperty.

PropertyValueToWideText (LPVOID pvDataBuffer, DWORD dwDataType, DWORD dwBufferSize, LPWSTR wszPropertyValue, DWORD dwTextCount)
Convert an arbitrary value to a wide string. The first three parameters are inputs defining the data to be converted (the location of the data is identified in *pvDataBuffer*, the type of data is identified in *dwDataType*, and the size of the buffer is identified in *dwBufferSize*). The last two parameters describe the output string: the string *wszPropertyValue* and the total number of wide characters *dwTextCount* in the *wszPropertyValue* buffer (not its size).
Returns: E_MM_DBNOTOPEN if data base not supplied in the constructor.
E_MM_INVALIDARG if *pvDataBuffer* or *wszPropertyValue* is NULL.

PropertyWideTextToValue (LPCWSTR wszPropertyValue, DWORD dwDataType, LPVOID pvDataBuffer, LPDWORD pdwBufferSize)
Convert a wide string to an arbitrary value. *wszPropertyValue* is the string to be converted and *dwDataType* is the data type to return in *pvDataBuffer*. *pdwBufferSize* is the initial size of *pvDataBuffer* in bytes, and on return it contains the actual size of the returned data.
Returns: E_MM_DBNOTOPEN if data base not supplied in the constructor.
E_MM_INVALIDARG if *pvDataBuffer* or *wszPropertyValue* is NULL.

PropertyValueToText (LPVOID pvDataBuffer, DWORD dwDataType, DWORD dwBufferSize, LPTSTR szPropertyValue, DWORD dwTextCount)
Convert an arbitrary value to a multi-byte string. The first three parameters are inputs defining the data to be converted (the location of the data is identified in

pvDataBuffer, the type of data is identified in *dwDataType*, and the size of the buffer is identified in *dwBufferSize*). The last two parameters describe the output string: the string *szPropertyValue* and the total number of wide characters *dwTextCount* in the *szPropertyValue* buffer (not its size).

Returns: E_MM_DBNOTOPEN if data base not supplied in the constructor.
E_MM_INVALIDARG if *pvDataBuffer* or *szPropertyValue* is NULL.

PropertyTextToValue (LPCTSTR szPropertyValue, DWORD dwDataType, LPVOID pvDataBuffer, LPDWORD pdwBufferSize)

Convert a wide string to an arbitrary value. *szPropertyValue* is the string to be converted and *dwDataType* is the data type to return in *pvDataBuffer*. *pdwBufferSize* is the initial size of *pvDataBuffer* in bytes, and on return it contains the actual size of the returned data.

Returns: E_MM_DBNOTOPEN if data base not supplied in the constructor.
E_MM_INVALIDARG if *pvDataBuffer* or *szPropertyValue* is NULL.

ImediaIndexPropertyValidation

Interface used to pass property validation parameters between an application and the index(es). All validation parameters are optional. Validation parameters may be supplied in the original scheme, or added/changed programmatically. Validation parameters are:

A lowest valid value - Combines with highest value to define a valid range.
A highest valid value - Combines with lowest value to define a valid range.
A list of valid values - Mutually exclusive with "lowest" and "highest" values.
A maximum count of characters (for text-based properties only).
A flags word. Bits are defined in the PROPVALID enum, described below.
Currently, there are two: the "unique" flag and the "displayname" flag. Unique means that the property value must be unique for all instances of the owning object. Displayname means that the property can be used to display an identifying name for the owning object.

```
typedef enum
{
   propvalid_Nil              =-1,          //invalid value
   propvalid_None             =0,           //No flags enabled
   propvalid_FlagUnique       =0x00000001,  //Value must be unique
   propvalid_FlagDisplayName  =0x00000002,  //field is the displayname (must be string)
} PROPVALID;
```

GetValidLo (VARIANT *pvarLo)

Get the lowest valid value, if any, for the property identified by the instance of the object implementing the IMediaIndexPropertyValidation interface returned as *pvarLo.*

PutValidLo (VARIANT varLo)

Set the lowest valid value for the property identified by the instance of the object implementing the IMediaIndexPropertyValidation interface as *varLo.* Pass an empty variant (vt == VT_EMPTY) to NULL out this value.

GetValidHi (VARIANT *pvarHi)

Get the highest valid value, if any, for the property identified by the instance of the object implementing the IMediaIndexPropertyValidation interface returned as *pvarHi.*

PutValidHi (VARIANT varHi)

Set the highest valid value for the property identified by the instance of the object implementing the IMediaIndexPropertyValidation interface as *varHi.* Pass an empty variant (vt == VT_EMPTY) to NULL out this value.

GetValidList (VARIANT **ppVar, LPDWORD pcVar)

Retrieves the list of valid values. List is returned as an array of VARIANT's in **ppVar*, with the array size in **pcVar.*

PutValidList (VARIANT *pVar, DWORD cVar)

Sets the list of valid values. List is passed as an array of VARIANT's in *pVar*, with the array size in *cVar.* Overrides any existing list.

GetCchValid (LPDWORD pcch)

Gets the maximum length *pcch* of a string parameter, if any.

PutCchValid (DWORD cch)

Sets the maximum length of a string parameter to *cch.*

GetFlags (LPDWORD pFlags)

Gets the flags word returne din *pFlags.* Flags are defined in PROPVALID enum.

PutFlags (DWORD flags)

Sets the value of the flags word as *flags*. Flags are defined in PROPVALID enum.

ImediaIndexSearchSink

Interface used for callbacks while a search is underway. Searches are asynchronous, and return each match as it is found. If you never call Search(), this interface will never be called.

OnSearchStarted ()

Called at the start of a search, to allow the client to perform client-specific actions.

OnObjectFound (REFGUID guidObjectType, DWORD dwInstanceID, REFGUID guidPropertySet, DWORD dwPropertyNumber, DWORD dwScore)

Called to return the values of each matched object. Indicates the object and property that matched (as well as the property set to which the property belongs), plus a score rating the "goodness" of the match, on a 0 – 10000 scale. The parameters are:

guidObjectType: the type of object that matched.

dwInstanceID: which object of type *quidObjectType* matched.

guidPropertySet: property set of the property (*dwPropertyNumber*) that matched the search string.

dwPropertyNumber: the property that matched the specified search string.

dwScore: the score of this particular match, on a 0 – 10000 scale, where 10000 equals 100%.

OnProgress (DWORD dwPercent)

If the client asked for "progress status" in the dwFlags word of the Search() API, this interface will be called at least once. It is called every time the "percent done" value changes from the value previously passed to the client. Values returned are in the range 0 – 100.

This interface also supports an additional efficiency feature. If you asked for "progress status" and the search is likely to be very short, we will not return a zero value through this callback. Thus, you could choose not to display a progress bar at all if a zero value wasn't received, since it would probably just flash immediately up to 100%.

OnSearchCompleted (DWORD dwFound, HRESULT hr)

Called once when the search is finished. Returns the number of matches *dwFound* and the success/failure status of the search in *hr*.

Possible values of *hr* parameter input: S_MM_CANCEL if search cancelled by client.